NASA Technical Memorandum 104632

Particle–Mesh Techniques on the MasPar

Peter MacNeice, Clark Mobarry, and Kevin Olson

MARCH 1996

NASA Technical Memorandum 104632

Particle–Mesh Techniques on the MasPar

Peter MacNeice
Hughes STX
Greenbelt, Maryland

Clark Mobarry
NASA/Goddard Space Flight Center
Greenbelt, Maryland

Kevin Olson
George Mason University
Fairfax, Virginia

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

1996

This publication is available from the NASA Center for AeroSpace Information,
800 Elkridge Landing Road, Linthicum Heights, MD 21090-2934, (301) 621-0390.

1 Introduction

Particle simulation techniques attempt to model many-body systems by solving the equations of motion of a set of particles or pseudo-particles which are used to represent the system [1]. Particle-In-Cell(PIC) techniques represent a popular variant on this theme, in which a numerical mesh is added to more efficiently compute the forces acting on these model particles. PIC codes are commonly used to model plasmas [2] and gravitational N-body systems.

Previous SIMD implementations of PIC [3, 4, 5] have reported serious performance degradation as the particles became more clustered in space. Indeed Walker[3] in his conclusions reported a suggestion that this performance degradation might make the Vlasov-Maxwell method more appropriate than PIC for plasma models on the CM-2.

In this report we consider three different approaches which we have tested, for implementing a 3D electrostatic plasma or gravitational PIC code on the SIMD architecture of the MasPar MP-2 at Goddard Space Flight Center. We show that it is possible, by combining algorithms to set a limit on the performance degradation resulting from particle clustering.

In the most abstract sense, the task we face in implementing our PIC code on the MP-2 is to map both an algorithm and a data structure to the architecture. The four basic steps in a PIC algorithm are

1. assign particle charge(mass) to the mesh
2. solve for the force field on the mesh
3. interpolate force from the mesh to the particle positions
4. push the particles.

In combination these four steps involve computation and communication between two different data structures. The field data has the qualities of an ordered array in the sense that each element has specific neighbors. The particle data has the qualities of an unstructured vector, in which element i refers to particle i, and no element has any special relationship to its neighbors in the vector.

Steps 2 and 4 are parallelizable in rather obvious ways, since they involve only simple and completely predictable data dependencies, and do not couple

the two data structures. Steps 1 and 3 however do couple the two data structures, with complicated and unpredictable data dependencies which evolve during the simulation. It is these steps which invariably dominate the execution times of parallel PIC codes, and will be the focus of this paper.

On a serial machine our code will execute its computational workload in a time which is independent of any correlations in the spatial locations of the particles. This is not true on parallel machines. Particle clustering can create communication and/or computational hot-spots which impair performance. For example, an algorithm which works very efficiently for an homogeneous plasma application may be very inefficient for a highly clustered gravitational N-body problem. This can be an important factor in choosing an algorithm.

In section 2 we describe the MasPar MP-2, in section 3 we define the computational tasks to be accomplished, in section 4 we outline the three algorithms tested, and in section 5 we present performance results for these algorithms.

2 The MasPar MP-2

The MasPar MP-2 at Goddard Space Flight Center has a SIMD architecture with 16384 processors. The nominal peak performance is 6.2 GFLOPS. Each processor has 64 KBytes of dedicated data memory. The processors are arranged in a 2D array with dimensions 128×128. Straight-line connections, known collectively as the X-net, exist between processors in the north, south, east, west, north-east, south-east, south-west and north-west directions. At the edges of the processor array the X-net wraps around so that the array has the same topology as the surface of a torus. Inter-processor communications can be achieved in one of two ways. The global router can be used for more complex patterns or for communication between widely spaced processors, while for regular patterns over short distances the X-net communications are much more efficient. The MasPar series broadens the definition of SIMD in at least one important way. It enables indirect addressing within a processor memory.

3 The Computational Task

Step 1 of the PIC process, the assignment of the particle charge (or mass) to the mesh to compute a density in each mesh cell is a scatter-with-add operation. The following Fortran 77 code will sum the charge density of the particles, using Nearest Grid Point(NGP) interpolation, and assuming mesh cells with sides of length h, and individual particle charge q.

```
parameter( n= ...)                    ! number of particles
parameter( nx=..., ny=..., nz=... )   ! dimensions of mesh
real    x(n), y(n), z(n)              ! particle coordinates
real    density(nx,ny,nz)             ! mesh charge(mass) density
        .
        .
        .

do i = 1, n                           ! loop through particle list

    ix = int( x(i)/h ) + 1
    iy = int( y(i)/h ) + 1
    iz = int( z(i)/h ) + 1

    density( ix, iy, iz ) = density( ix, iy, iz) + q/h**3

end do
```

On a serial machine this loop executes in order. On a parallel machine however (and also on vector processors) the operations within the loop can be executing for many different values of i simultaneously. On the MasPar the particle data (vectors x,y,z) and the density array, density, are distributed across the memory of the processors in the processor array. At a given time up to 16,384 processors might be simultaneously calculating the contributions of different particles to the array density. Each of these contributions needs to be communicated from the processor storing that particle data to the processor storing the appropriate element of density. This task is complicated by the possibility that more than one, and possibly a great

many particles might be simultaneously trying to update the same element of density. In the presence of these communication hot spots, we need to guarantee that no contributions are ignored, and that the sums are accumulated efficiently.

Step 3, the interpolation of force from the mesh to the particle locations is a gather operation. For the simple NGP scheme, in Fortran 77,

```
real  particle_force_x( n)          ! x component of force on particles
real  mesh_force_x( nx, ny, nz)     ! x component of force at mesh points

      .
      .
      .

do i = 1, n                         ! loop through the particle list

     ix = int( x(i)/h ) + 1
     iy = int( y(i)/h ) + 1
     iz = int( z(i)/h ) + 1

     particle_force_x( i) = mesh_force_x( ix, iy, iz)
     particle_force_y( i) = mesh_force_y( ix, iy, iz)
     particle_force_z( i) = mesh_force_z( ix, iy, iz)

end do.
```

Once again we can see that on the MasPar this task can suffer from communication hot spots, when many particles are physically located in the same mesh cell.

These then are the tasks we wish to implement on the MasPar. For the sake of clarity, we have illustrated these tasks using simple Nearest Grid Point interpolation. However NGP is rarely ever used in PIC codes, and so instead we will use Cloud-in-Cell. We assume that each particle is a uniform density charge(mass) cloud in the shape of a cube with sides of length h. In 3D CIC each particle overlaps 8 mesh cells and so during the charge assignment tasks

it makes 8 separate contributions to the charge density. Similarly the force on a particle is obtained by interpolation from the force at the 8 mesh points bounding the particle's location.

4 Charge Assignment and Force Interpolation Algorithms

On a distributed memory machine, such as the MasPar, data layout across processor memory is an integral part of algorithm design. The challenge on the MasPar is to spread the computation and communication workload as evenly as possible, while minimizing the amount of global router communication required.[1]

The field arrays will be laid out so as to optimize the field solver routine. We do not need to consider the fine details of this layout, which will vary depending on the exact size of the physical mesh and the size of the processor array. All we really need to recognize is that any acceptable layout will establish a mapping between physical mesh points and the processor array so that it includes most if not all the processors, and that nearest neighbor mesh cells will map to processors which are no further apart than nearest neighbors. For example if we have a 3D mesh of size $128 \times 128 \times 128$ and a processor array of size $N_{proc} = 128 \times 128$, we could map cell (i, j, k) into processor (i, j), rendering the third dimension in processor memory.

In the discussion that follows we will assume this exact problem size, processor array size and field data layout. The solution of Poisson's equation for the electricstatic potential for this mesh size, using MasPar library fft routines, takes 1 second. We will also assume cloud-in-cell(CIC) weighting for both charge assignment and force interpolation.

The major design question which faces us is how to distribute the particle data. There are some obvious choices which focus either on computational load balance or on efficient interprocessor communication.

[1] A plural floating point multiply takes 40 clocks on the MP-2, an X-net operation sending a real number a distance of 1 processor takes 41 clocks, and a random communication pattern using the global router, with all processors participating takes $\sim$ 5000 clocks.

4.1 Uniform Load Balance - with communication hot spots

The first option is to parcel the particles out evenly amongst the processors, paying no attention to their physical locations[4]. We have implemented this in both MPFortran and MPL.[2] This particle assignment option achieves the best computational load balance during the particle push and during the purely computational parts of the charge deposition and force interpolation tasks. It also makes memory management easier, since we know exactly how much memory we will need in every processor.

However it makes very heavy use of the global router for interprocessor communication. Any given particle can potentially seek to deposit charge on any of the processors in the processor array. Using CIC for a 3D model, each particle has 8 charge contributions to distribute to a $2 \times 2 \times 2$ block of elements somewhere in the charge array. In the MPFortran implementation each of these 8 words represents a distinct message to be sent via the router. In MPL we can pack these 8 components into a message which is then sent by the global router to one of the processors storing the $2 \times 2 \times 2$ block, which then distributes them, as required among its neighboring processors using the X-net.

Similarly during the force interpolation the particle needs to interpolate between the 24 field components associated with the same $2 \times 2 \times 2$ block (ie 8 components for each of the x, y and z directions). Again the MPFortran gathers these 24 data words using 24 separate messages collected via the global router. In MPL we have more direct control over the communication process. In MPL it is more efficient to pack the 24 data words into a single message. Using Xnet calls, each virtual processor pre-packs the 24 words which will be needed by any particle whose low index corner lies within its mesh cell. Then we use the router to fetch the appropriate pre-packaged 24 words for each particle, after which all the force interpolations can be done on processor.

This scheme is expected to be slow because of its extensive use of the global router, and it scales poorly in situations where clustering occurs. Communication hot spots occur when a large number of messages are being sent

[2]MPL which is an abbreviation for MasPar Programming Language is a parallel dialect of C.

to the same processor at the same time. The processors can only process one router message at a time. If at a given time n^p particles are physically located in cells which map into processor p, then processor p will need to receive n^p messages during that timesteps charge deposition. This algorithm therefore will scale as n^p_{max}, the maximum value of n^p across the processor array.

The MPFortran implementation of this algorithm is essentially an optimized version of the sendwithAdd function from MPL. It can be obtained in MPFortran through the use of the MPFortran compiler 'collisions' directive. For example, the charge assignment code listed in section 3 would become

```
real    x(n), y(n), z(n)              ! particle coordinates
real    density(nx,ny,nz)             ! mesh charge(mass) density
integer ix(n), iy(n), iz(n)
        .
        .
        .

     ix = int( x/h ) + 1
     iy = int( y/h ) + 1
     iz = int( z/h ) + 1

cmpf collisions
     density( ix, iy, iz ) = density( ix, iy, iz) + q/h**3.
```

4.2 A Particle Migration Strategy

We can avoid the router completely if we distribute the particles amongst the processors according to the same mapping used for the field arrays [3][6].

If a particle lies in cell (i, j, k) we store it on the processor to which we mapped cell (i, j, k). During the charge deposition, no particle will need to send charge any further than to a neighboring processor, and during the force interpolation the mesh field values which the particle needs are either on processor or stored by a neighbor. This enables us to use X-net communications exclusively. To maintain this locality we are required to migrate particle information from processor to processor as the particles move between mesh cells. Since our timestep constraint limits the distance any particle can travel

during that timestep to less than one mesh cell width, the migration can be achieved efficiently using the X-net.

There are of course drawbacks associated with this scheme. The additional code needed to perform the particle data migration makes the algorithm more difficult to program and debug. It also suffers from load imbalance as particle clustering develops. In this case both the communication and computation costs scale as n^p_{max}. Processor memory management is tricky. We have to allocate enough memory that the most heavily populated processor does not run out of memory. However this means that a lot of memory space in other processors will be allocated and never used.

One possible solution to the memory management weakness is to supplement this scheme with a backup routine similar to that used in the uniformly load balanced technique above. We will refer to this as the 'hybrid migration' scheme. Two distinct particle populations are identified, those which have been migrated successfully(population I) and those which have not(population II). We use the migration strategy wherever possible. Any population I particle which tries to migrate to a processor whose memory is already full will be left where it is and relabeled as population II. After we deposit the population I charge to the mesh, we use the global router to deposit charge from any population II particles. Similarly, when we have completed force interpolation for the population I particles we use the router approach to find the forces for any population II particles. At regular intervals (i.e. every 10 time steps perhaps), we test to see if the population II particles might now be placed into the correct processors and so transferred back to population I. This hybrid scheme enables us to minimize memory wastage. Its performance depends on the spatial distribution of particles, and lies somewhere between that of the pure migration approach and of the load balanced approach.

4.3 An Algorithm without Communication Hot Spots

We can eliminate communication hot spots and achieve a more attractive scaling of the charge deposition and force interpolation tasks by combining messages to the greatest extent possible before using the global router hardware.

This is possible if, before each timestep, we sort the particles so that all particles that contribute to the same mesh location form a contiguous

segment in the particle list. This implies that there is at most one segment in the particle list corresponding to each mesh location. Then we can use a segmented vector scan-add to sum the charge contributions in each segment. The accumulated charge density sums can then be distributed, with minimal router contention, to the appropriate processors. The force interpolation can be performed in a similar way, taking advantage of the sorted particle list to avoid communication hot spots. The particle at the tail of each segment fetches the force data that it needs. Since no two segment tails want the same data there is minimal router contention. This data is then copied from the segment tail to the other particles in that segment using a segmented vector scan-copy.

Efficient parallel algorithms exist for both the sorting and segmented vector scan operations.

To sort the particle list we used a version of the bitonic sort [7] which is available as share-ware [8]. This MPL coding of the bitonic sort executes in 0.63 seconds on a 16K processor MP-2 for n=1048576 64-bit records. An alternative sort, the B-Flashsort[9][10] with recursive sub-sampling promises to be faster when the number of particles grows over half a million on a 16K processor MP-2.

A segmented vector scan-add is essentially a fast method to compute linear recurrences such as:

```
parameter(n=16)
real seg(n),src(n),sum(n)
data seg /0,1,1,1,1,0,1,0,1,1,0,1,1,1,1,1/
data src /1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1/
do i=1,n
   sum(i) = seg(i)*sum(i-1) + src(i)
enddo
```

where `seg(1) = 0`, and where `seg(i) = 0` wherever `i` is the first index of a new segment. The results of this serial code snippet are

```
seg  =  0 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1
src  =  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
sum  =  1 2 3 4 5 1 2 1 2 3 1 2 3 4 5 6.
```

This operation is not a serial bottleneck. It can be coded to execute in $O((n/p)\log_2 \delta p)$ SIMD operations on the MasPar MP-2, where n is the number of elements, p is the number of processors and δp is the number of pro-

cessors spanned by the longest segment [11]. Our parallel MPL coding of the recurrence algorithm above executes in 2.8 microseconds for n=1048576 records. Our implementation of this algorithm[12] uses a data layout in which the n particles in the sorted particle list are stored in blocks of size n/p with the first block assigned to processor 1, the second to processor 2, and so on.

We note that this strategy has a natural advantage for implementation of the particle-mesh(PM) part of a particle-particle-particle-mesh(P^3M) code [1]. Particle sorting is required in the particle-particle(PP) section of the force evaluation to establish which particles are near neighbors. If the sort/scan-add scheme was used for the PM parallelization no further sorting would be required during the PP stage.

5 Performance

We have explored the performance of these algorithms as a function of both the degree of spatial clustering of particles and of the total number of particles.

5.1 Spatial Clustering of Particles

As we have mentioned, some of these algorithms will suffer from computational and/or communication hot spots when spatial clustering of particles occurs.

To illustrate this we have developed the following idealized test case. n particles have been placed into n_{cl} clusters, whose centers are located at randomly chosen coordinates within the 128^3 mesh. Each cluster contains n/n_{cl} particles. The clusters have a radius R given by

$$\frac{R}{\Delta} = 128 \times n_{cl}^{-1/3} f$$

and the particle locations within a cluster are chosen randomly, subject to an r^{-2} density distribution, where r is the particles distance from the center of its cluster. The factor f is used to adjust the radius of the clusters. This test case mimics the clustering which might be encountered modeling a small number of interacting galaxies. By increasing f sufficiently we can make the particle distribution completely random.

The load balanced(both MPFortran and MPL versions), hybrid migration, and sort/scan-add algorithms were tested. The clusters were assigned a randomly chosen translational velocity to ensure that in the hybrid migration case the particle migration routines were exercised.

We chose $n = 10^6$ and $n_{cl} = 10$, and adjusted the degree of clustering by varying f. Results are presented in figure 1. Figure 1 shows average execution times per timestep for the combination of charge deposition and force evaluation tasks for all 3 algorithms. These times are plotted against the time averaged value of n^p_{max}/n^p_{avg}. Recall that n^p is the number of particles sending charge to processor p, n^p_{max} is the maximum value of n^p amongst all the processors, and n^p_{avg} is the average value.

For the load balanced algorithm there are no computational hot spots. The algorithms efficiency falloff as f decreases and clustering develops is a direct consequence of the communication hot spots which develop. The MPFortran and MPL versions overall times and scaling were very similar but with different divisions between the charge deposition and force evaluation tasks. The MPFortran compiler generates more efficient code but MPL provides finer control with which to optimize interprocessor communications.

For the hybrid migration algorithm the situation is more complicated. Let us assume that each processor's memory is large enough to accommodate n_{max} particles. Let us also assume for argument sake that $n_{max} = 2n/128^2$. For a uniform spatial distribution of particles, ie $f = \infty$, all the processors have the same number of particles, all particles can be designated population I and can be migrated successfully, and there are no computational or communication hot spots. As we reduce f clustering starts to develop. The computational and communication balances deteriorate as f decreases. As we decrease f further, we find that some particles can no longer be migrated successfully and are classified as population II. This does not make the computational load balance any worse, but the communication load balance continues to deteriorate. As f continues to decrease the fraction of particles in population II grows steadily, until eventually almost all the particles are population II. At this point the overhead associated with maintaining two populations and dealing with the few population I particles remaining is no longer cost effective, and the load balanced scheme is faster.

The sort/scan-add algorithm has a much more attractive scaling as clustering develops. The current implementation of the bitonic sort and the scan-add algorithms have deterministic execution times for a given number

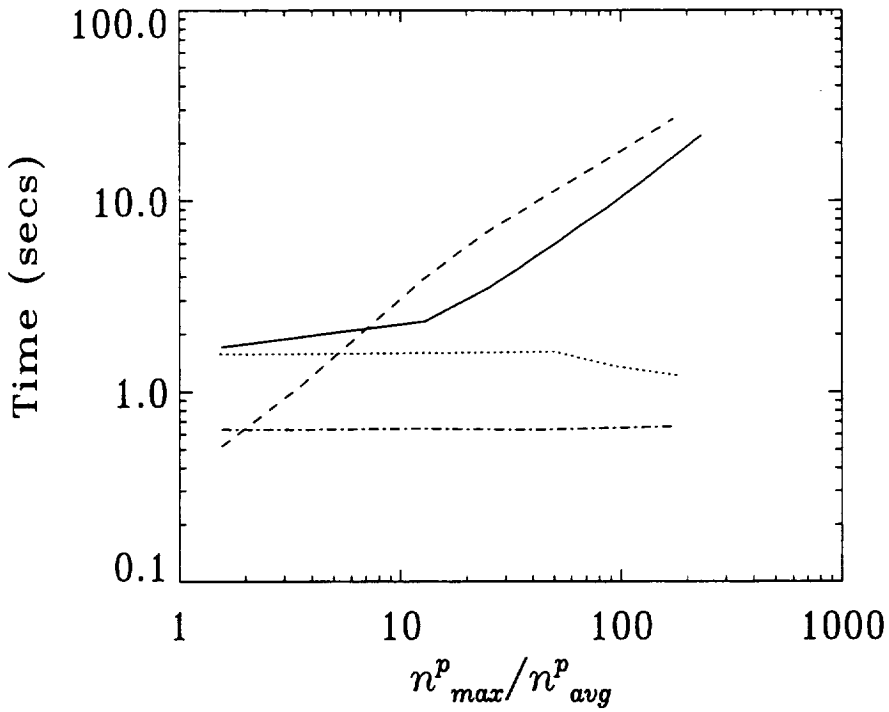

Figure 1: Average execution time per timestep for the combination of charge deposition and force evaluation tasks for the MPFortran load balanced(solid), sort/scan-add(dotted) and MPL hybrid migration(dashed) algorithms as a function of the degree of clustering. The dot-dash line shows the times for the same tasks on one processor of a Cray C90.

of particles and processors. However in Figure 1 the execution time decreases with increasing clustering. This is because the global router is used in three ways: (1) to move the particles into position after the bitonic sort found the new ranking of the particles; (2) to deposit the charge from the particle segment tails to the mesh; and (3) fetch the force interpolation coefficients from the mesh to the head of the particle segments. When the clustering increases, for a fixed total number of particles, the number of distinct segments in the sorted particle list decreases. Thus as the clustering increases, we observe the global router traffic and the execution time decreasing. The particle migration into a sorted list is the most expensive component of this algorithm comprising 55% of the execution time. If a more efficient sort than the

bitonic sort is implemented, then the sort/scan-add algorithm would become still more competitive.

For low degrees of clustering the migration strategy is faster but as clustering increases the superior scaling properties of the sort/scan-add approach win through.

The most significant implication of these scaling curves is that we can limit the performance degradation associated with spatial clustering. There is no reason why we could not combine the migration and sort/scan codes into a hybrid algorithm which measured the degree of particle clustering at regular intervals as the simulation ran and chose the more efficient algorithm. Some additional work would be required to translate the data structures and manage the associated allocations and deallocations of memory anytime an algorithm switch was deemed advantageous. However this would not be costly since these switches would occur infrequently. In most cases the simulation progresses steadily from more uniform to more clustered and switching would be needed only once.

We repeated these calculations on one processor of a Cray C90 to provide a familiar point of reference for these execution times. The C90 timings are also plotted in figure 1. We relied on a built-in feature of the cf77 compilation system to vectorize the charge deposition task. With appropriate command line arguments the compiler executes the charge deposition loop over the particle list in blocks whose size is set to be the length of the vector pipeline. Before each block, it inserts a test using a vectorized scatter and a gather on integer vectors to determine if any dependencies exist in that blocks scatter pattern. If it finds none, then it vectorizes that block. Otherwise it executes that block serially. We also shuffled the particle list after the particles were loaded into memory, to ameliorate bad memory bank contention which is likely to become more serious as clustering increases. The timings reported are averages over 50 timesteps. The hardware performance monitor(HPM) reported 338 - 347 Mflops for these runs, with an average vector length of 116 and approximately 0.6 memory conflicts per reference.

5.2 Scaling With Total Particle Number

Spatial clustering is just one factor which will influence the relative performance of these 3 algorithms. Another important scaling is as a function of the total number of particles. Different schemes will incur different setup

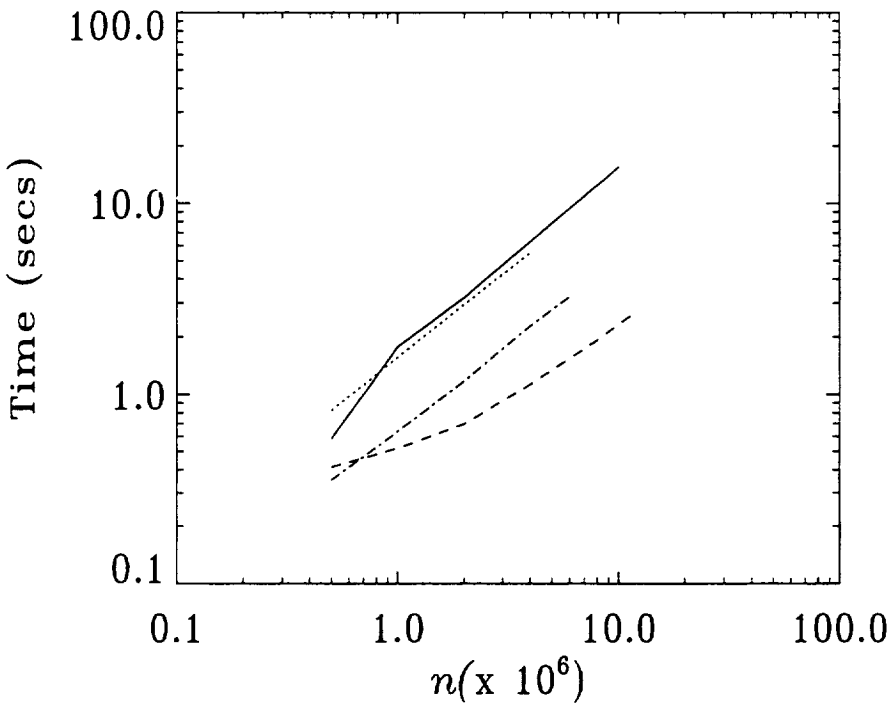

Figure 2: Average execution time per timestep for the combination of charge deposition and force evaluation tasks for the MPFortran load balanced(solid), sort/scan-add(dotted) and MPL hybrid migration(dashed) algorithms as a function of the total number of particles for a random spatial distribution. The dot-dash line shows the corresponding times for one processor of a Cray C90.

overheads each timestep, ie initializing buffer arrays, pre-sorting particle lists etc, which may be amortized more effectively for larger numbers of particles.

To explore this we ran all 3 algorithms for a range of particle numbers, with the particles distributed randomly in space, and with a Maxwellian velocity distribution which resulted in an average of 16% of the particles changing mesh cell every timestep. Results are plotted in figure 2.

The load balanced and sort/scan algorithms scale linearly with total particle number as expected. The hybrid migration cost grows more slowly for smaller particle numbers but starts to approach linear scaling at large particle numbers. One reason for this is that the load imbalance for this random

distribution decreases as $1/\sqrt{n^p_{avg}}$. It should be pointed out that figure 2 is plotted for a random distribution which strongly favors the hybrid migration schemes.

6 Conclusions

We have studied the performance of three different parallel algorithms for the particle charge deposition and force evaluation tasks of a 3D electrostatic or gravitational PIC code on the MasPar MP-2. We considered how the algorithms scale as a function of particle clustering and with the total number of particles. The best choice of algorithm depends on the degree of spatial clustering of the particles and on the importance of ease of programming to the programmer.

The sort/scan approach, which minimizes message traffic between the particle and mesh data structures, is clearly superior when any significant spatial clustering occurs. Therefore for gravitational N-body applications this is the recommended choice. When the spatial clustering of particles is weak, as in many electrostatic plasma applications the hybrid particle migration strategy is fastest.

However both these schemes are significantly more complex to code and debug than the MPFortran coding of the load balanced algorithm. If ease of programming is important and the particles are not severely clustered then this approach should be considered.

We have shown that the influence of particle clustering on the performance of our PIC code on the MasPar MP-2 can be managed effectively by combining the migration and sort/scan approaches, and we can see no reason why this conclusion should not apply to SIMD architectures in general.

References

[1] R.W. Hockney and J.W. Eastwood, Computer Simulation Using Particles, Institute of Physics, (1988).

[2] C.K. Birdsall and A.B. Langdon, Plasma Physics via Computer Simulation, McGraw-Hill Inc., (1981).

[3] D.W. Walker, "Particle-in-Cell Plasma Simulation Codes on the Connection Machine", *Computing Systems in Engineering*, **2**, 307, (1991).

[4] R.G. Hohlfeld, N.F. Comins, D. Shalit, P.A. Shorey and R.C. Giles, "Implementation of Particle-in-Cell Stellar Dynamics Codes on the Connection Machine-2", *The Journal of Supercomputing*, **7**, 417, (1993).

[5] C.S. Lin, A.L. Thring and J. Koga, "A Parallel Particle-in-Cell Model for the Massively Parallel Processor", *Journal of Parallel and Distributed Computing*, **8**, 196, (1990).

[6] P. MacNeice, "An Electro-magnetic PIC Code on the MasPar", *Proc. of the sixth SIAM conf. on Parallel Processing for Scientific Computing*, 129, (1993).

[7] K. Batcher, "Sorting Networks and Their Applications", *Proceedings of the AFIPS Spring Joint Computing Conference*, **32**, p. 307, (1968).

[8] J. Prins, "Efficient Bitonic Sorting of Large Arrays on the MasPar MP-1", UNC Dept. of Computer Science TR91-041, 1991. PostScript (119K) and distribution (17K) (shar file) at `http://www.cs.unc.edu/~prins/`.

[9] W. Hightower, J. Prins, J. Reif, "Implementations of Randomized Sorting on Large Parallel Machines", Proc. 3rd Symposium on Parallel Architectures and Algorithms, ACM, 1992.

[10] J. Prins, "B-Flashsort: A High-performance parallel sort for the MasPar MP-1 and MP-2", UNC Dept. of Computer Science TR92-091, 1992. PostScript (113K) and distribution (44K) (shar file) at `http://www.cs.unc.edu/~prins/`.

[11] G.E. Blelloch, Vector Models for Data-Parallel Computing, MIT Press, (1990).

[12] C. Mobarry, et al, in preparation, (1995).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1996	Technical Memorandum

4. TITLE AND SUBTITLE

Particle–Mesh Techniques on the MasPar

5. FUNDING NUMBERS

Code 934

6. AUTHOR(S)

Peter MacNeice, Clark Mobarry, and Kevin Olson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Hughes STX
4400 Forbes Boulevard
Lanham, Maryland 20706

8. PERFORMING ORGANIZATION REPORT NUMBER

96B00052

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Goddard Space Flight Center
National Aeronautics and Space Administration
Washington , DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

TM-104632

11. SUPPLEMENTARY NOTES

Peter MacNeice: Hughes STX, Lanham, Maryland
Kevin Olson: George Mason University, Fairfax, Virginia

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified-Unlimited
Subject Catagory: 75
This report is available from the NASA Center for AeroSpace Information, 800 Elkridge Landing Road, Linthicum Heights, MD 21090; (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

We investigate the most efficient implementations of the charge (mass) assignment and force interpolation tasks of a particle-in-cell code on the SIMD architecture of the MasPar MP2. Three different approaches were tested. The first emphasized uniform computational (not necessarily communication) load balance and ease of programming. The second exploited the speed of the Xnet interprocessor communication network using a particle data migration strategy. The third used sorting and vector scan-add operations on the particle dataset to minimize the communication traffic required between the particle and mesh data structures. Algorithm efficiencies were measured as a function of the degree of spatial clustering of the particles, and as a function of the total number of particles. The sort/scan-add strategy gave the best performance for a broad range of degree of spatial clustering. It was only beaten by the migration strategy in the regime of weak clustering. Our results indicate how a hybrid algorithm combining the migration and sort/scan-add strategies can set an upper limit on the performance degradation associated with the spatial clustering of particles.

14. SUBJECT TERMS

Particle–Mesh, SIMD Programming

15. NUMBER OF PAGES

22

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SPECIAL FOURTH-CLASS RATE
POSTAGE & FEES PAID
NASA
PERMIT No. G27